Report on the Mini Project

Topic: Car Rental System

Team 12

Khúc Ngọc Nam – 20230086

Lê Trung Dũng – 20235489

Mai Anh Quân – 20235547

Ngô Văn Đông – 20235486

Nguyễn Đức Mạnh – 20235525

Nguyễn Vương Trung Hiếu - 20235496

Course: Object-oriented Programming 156626 – 2024.1
Dr. Tran The Hung
14/12/2024

ACKNOWLEDGEMENT

To finish this project, we have received many helps from people. We would like to express my sincere thanks and gratitude to all these respected people.

First, we are thankful to Dr. Tran The Hung for letting us work on this project as well as her enthusiastic support and detailed guidance in completing this project. We got to learn a lot more about this project especially in OOP technique and Java language which will be very helpful for us not only now abut also in the future.

Moreover, we would like to thank some of our classmates for their times and effort that help us answer some questions support our report.

Finally, we also need support from our ''AI Friends'', ChatGPT, which help us a lot with thing that we didn't know completely

1. INTRODUCTION

The **Car Rental System** is designed to streamline and manage car rental services efficiently. It provides a robust platform for administrators, customers, and car owners to interact seamlessly, ensuring a smooth workflow from vehicle registration to rental transactions. The system is intended for businesses or organizations that offer vehicles for rent, aiming to automate processes like managing car inventories, tracking rentals, user authentication, and account handling.

2. MINI PROJECT DESCRIPTION

The **Car Rental System** consists of several key components that work together to fulfill its core functionalities:

Admin:

- Admin users are responsible for managing accounts, adding or deleting customers and car owners, and viewing or saving account data.
- They play a supervisory role and ensure the system runs smoothly.

Customer:

- Customers are users who wish to rent cars.
- They can browse available cars, update their rental status, and interact with the car inventory to request vehicles.

CarOwner:

- Car owners register and manage their cars in the system.
- They can add or remove cars, save their cars to storage, and keep track of their registered vehicles.

Cart:

- The cart allows users to temporarily store selected cars before confirming a rental.
- It includes functions for adding/removing cars, calculating total costs, and viewing cars within the cart.

Car:

- This class represents individual cars, including attributes such as license plates, brand, type, rental price, and manufacturing year.
- Cars are managed by car owners and rented by customers.

Store:

- The store manages the inventory of cars using a HashMap for efficient tracking.
- It allows for adding, removing, and displaying cars in the rental system.

Person (Abstract Class):

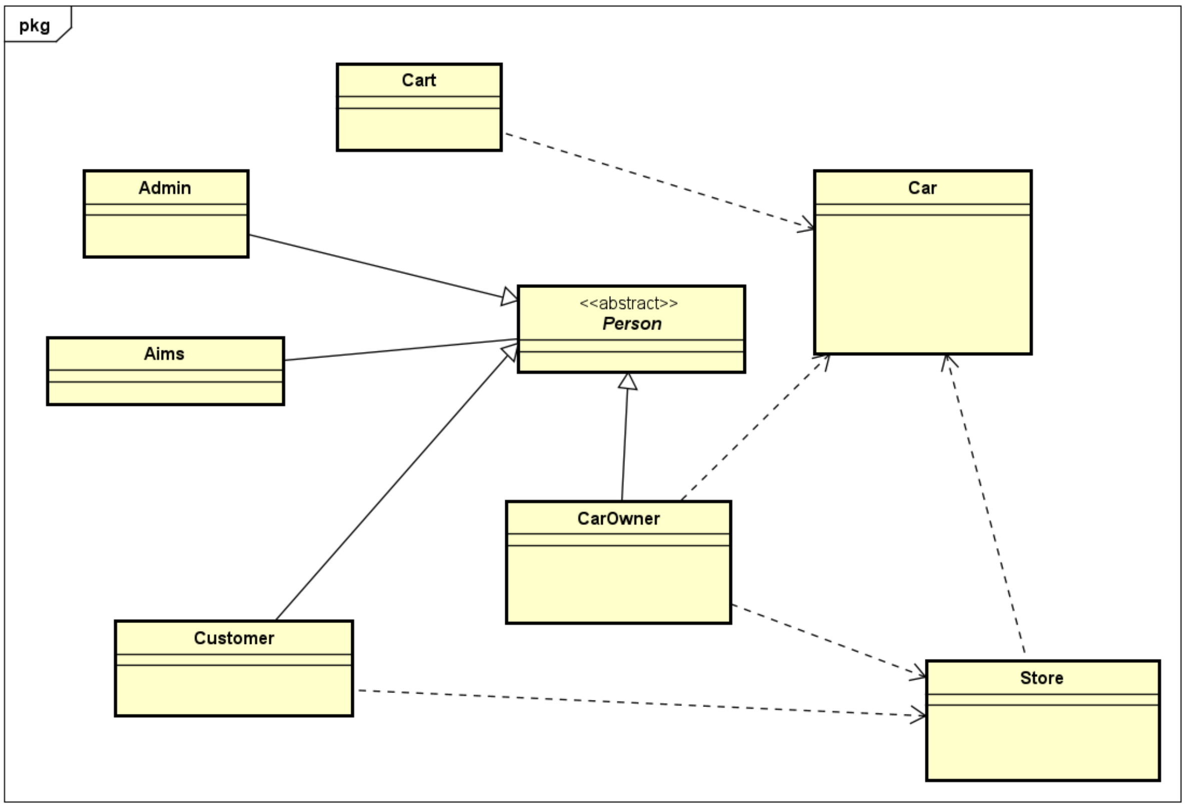
- The abstract Person class provides foundational attributes and behaviors (username, password, login functionality) shared by all users in the system, including admins, customers, and car owners.

Aims:

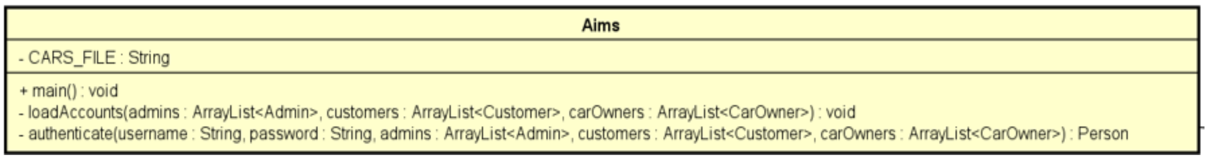
- This class functions as the main entry point for the system, handling data initialization, account loading, and user authentication.

3. DESIGN

3.1. General class diagram



3.2. Aims



```
Aims
- CARS_FILE : String
+ main() : void
- loadAccounts(admins : ArrayList<Admin>, customers : ArrayList<Customer>, carOwners : ArrayList<CarOwner>) : void
- authenticate(username : String, password : String, admins : ArrayList<Admin>, customers : ArrayList<Customer>, carOwners : ArrayList<CarOwner>) : Person
```

Purpose: Serves as the main entry point of the program and handles account loading and user authentication.

Attributes:

- CARS_FILE: A constant string to specify the file path storing car-related data.

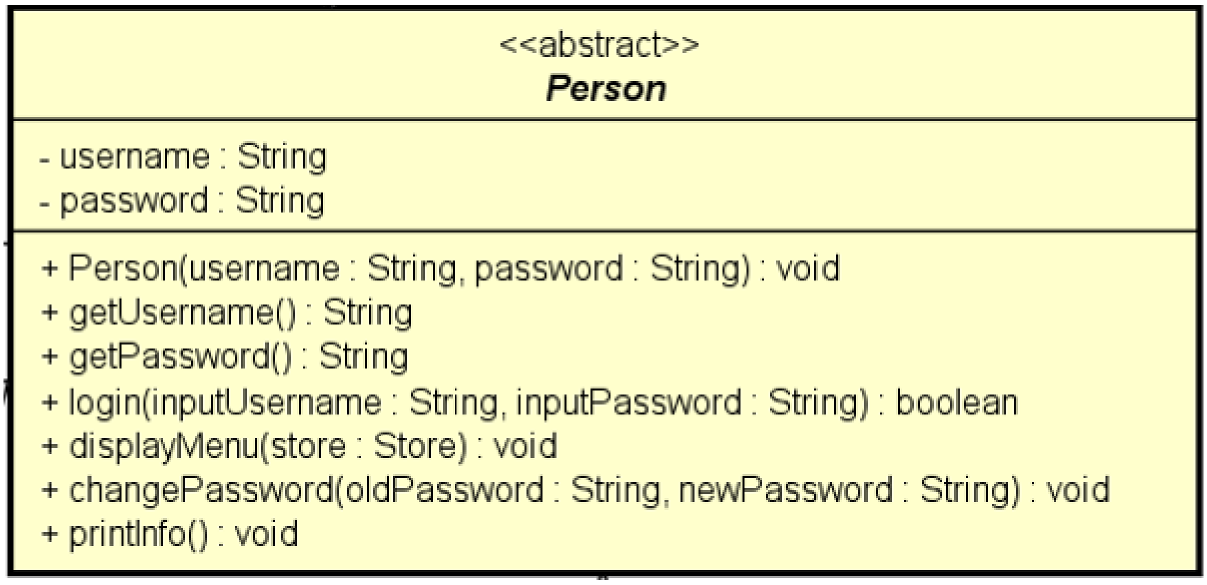
Methods:

+ main(): The main program execution starting point.

- loadAccounts(): Loads accounts (admins, customers, car owners) into memory.

- authenticate(): Authenticates users based on provided credentials.

3.3. Person (Abstract class)



<<abstract>>
Person
- username : String
- password : String
+ Person(username : String, password : String) : void
+ getUsername() : String
+ getPassword() : String
+ login(inputUsername : String, inputPassword : String) : boolean
+ displayMenu(store : Store) : void
+ changePassword(oldPassword : String, newPassword : String) : void
+ printInfo() : void

Purpose: Provides a general base class for Admin, Customer, and CarOwner classes.

Attributes:

- username: The login identifier.

- password: The password for authentication.

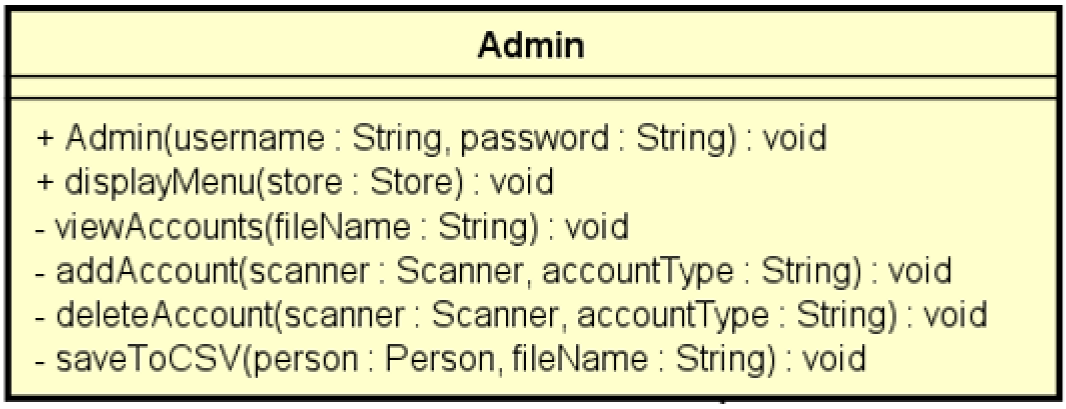
Methods:

+ Person(username, password): Constructor to initialize common attributes.
+ getUsername() and getPassword(): Accessor methods.
+ login(): Validates login credentials.
+ changePassword(): Allows updating the user’s password.

OOP Technique: Abstraction

- Defined as an abstract class with methods shared by multiple subclasses.

3.4. Admin (Inherit from person)


Admin
+ Admin(username : String, password : String) : void
+ displayMenu(store : Store) : void
- viewAccounts(fileName : String) : void
- addAccount(scanner : Scanner, accountType : String) : void
- deleteAccount(scanner : Scanner, accountType : String) : void
- saveToCSV(person : Person, fileName : String) : void

Purpose: Manages administrative tasks such as adding, deleting, and viewing user accounts.

Attributes:

- None explicitly defined.

Methods:

+ Admin(username, password): Constructor.
+ displayMenu(): Displays options available to an admin.

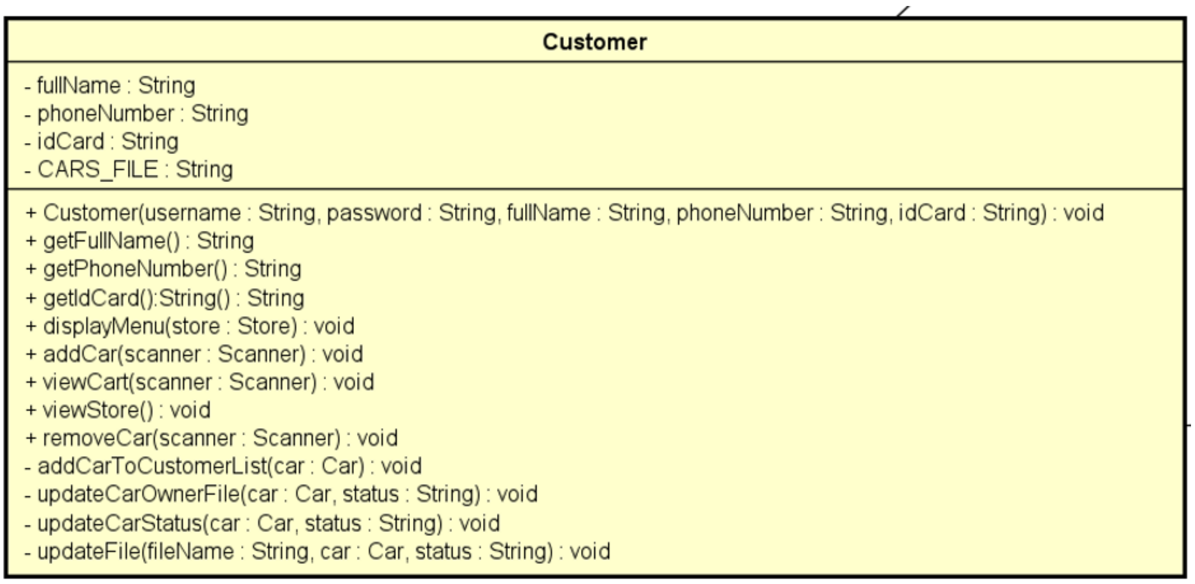
- viewAccounts(): Views all accounts stored in a file.

- deleteAccount(): Deletes specified user accounts.
- saveToCSV(): Saves accounts back to a CSV file.

OOP Technique: Inheritance

- Inherits from the **Person** class to reuse its attributes and methods.

3.5. Customer (Inherit from person)



```
Customer
- fullName : String
- phoneNumber : String
- idCard : String
- CARS_FILE : String
+ Customer(username : String, password : String, fullName : String, phoneNumber : String, idCard : String) : void
+ getFullName() : String
+ getPhoneNumber() : String
+ getIdCard():String() : String
+ displayMenu(store : Store) : void
+ addCar(scanner : Scanner) : void
+ viewCart(scanner : Scanner) : void
+ viewStore() : void
+ removeCar(scanner : Scanner) : void
- addCarToCustomerList(car : Car) : void
- updateCarOwnerFile(car : Car, status : String) : void
- updateCarStatus(car : Car, status : String) : void
- updateFile(fileName : String, car : Car, status : String) : void
```

Purpose: Represents a customer interacting with cars, like adding or removing cars.

Attributes:

- fullName: Full name of the customer.
- phoneNumber: Contact phone number.
- idCard: Customer's identification card.
- CARS_FILE: File path to store car-related records.

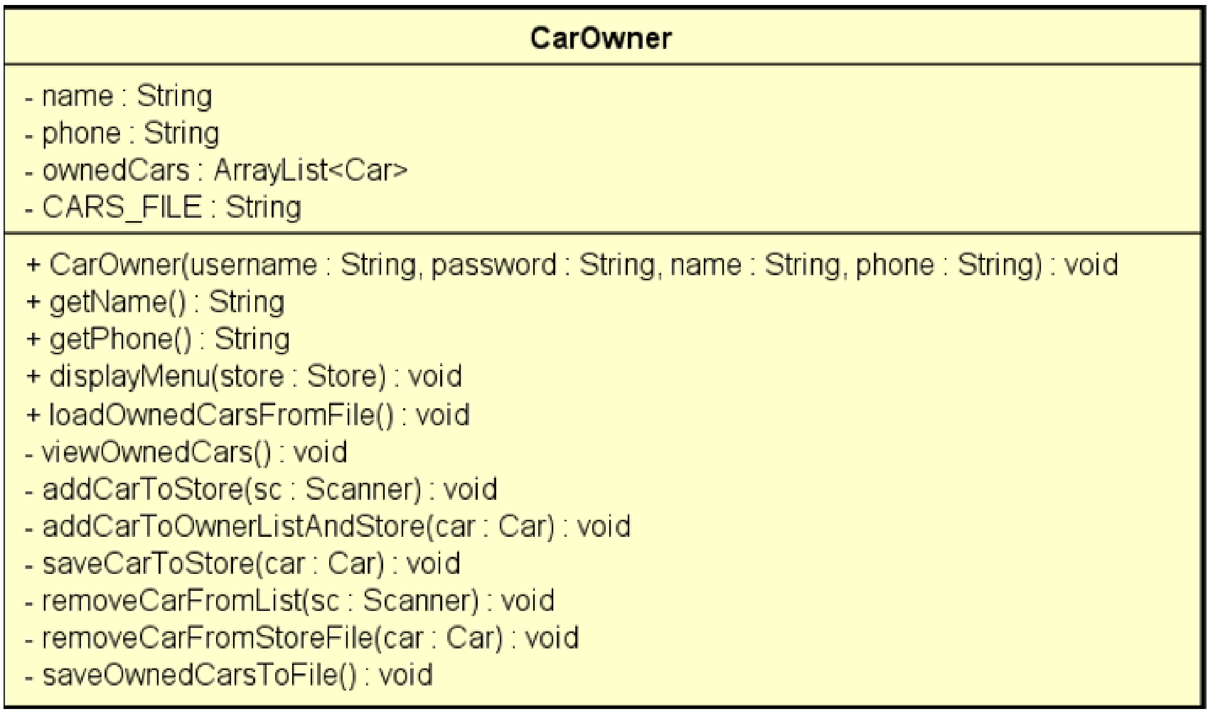
Methods:

+ Customer(username, password, fullName, phoneNumber, idCard): Constructor to initialize customer details.
+ getFullName(): Retrieves the full name of the customer.
+ getPhoneNumber(): Retrieves the phone number of the customer
+ getIdCard(): Retrieves the ID Card of the customer
+ addCar(): Adds cars to a customer's account.
+ viewCars() and removeCar(): Allows viewing and removing cars.
+ removeCar(): Removes a car associated with the customer.
- updateCarOwner(): Updates ownership details for a specific car.
- updateCarStatus(): Updates the status of a car (e.g., rented or available).
- updateFile(): Updates the persistent storage file with current car or customer data.

OOP Technique: Inheritance

- Inherits from **Person** to avoid redundancy.
- Encapsulation is also used through getters/setters.

3.6. Car owner (Inherit from person)



```
CarOwner
- name : String
- phone : String
- ownedCars : ArrayList<Car>
- CARS_FILE : String
+ CarOwner(username : String, password : String, name : String, phone : String) : void
+ getName() : String
+ getPhone() : String
+ displayMenu(store : Store) : void
+ loadOwnedCarsFromFile() : void
- viewOwnedCars() : void
- addCarToStore(sc : Scanner) : void
- addCarToOwnerListAndStore(car : Car) : void
- saveCarToStore(car : Car) : void
- removeCarFromList(sc : Scanner) : void
- removeCarFromStoreFile(car : Car) : void
- saveOwnedCarsToFile() : void
```

Purpose: Represents a car owner who can manage their cars and interact with the store.

Attributes:

- name: Owner's name.
- phone: Contact phone number.
- ownedCars: A list to store cars owned by the user.
- CARS_FILE: File path to store car data.

Methods:

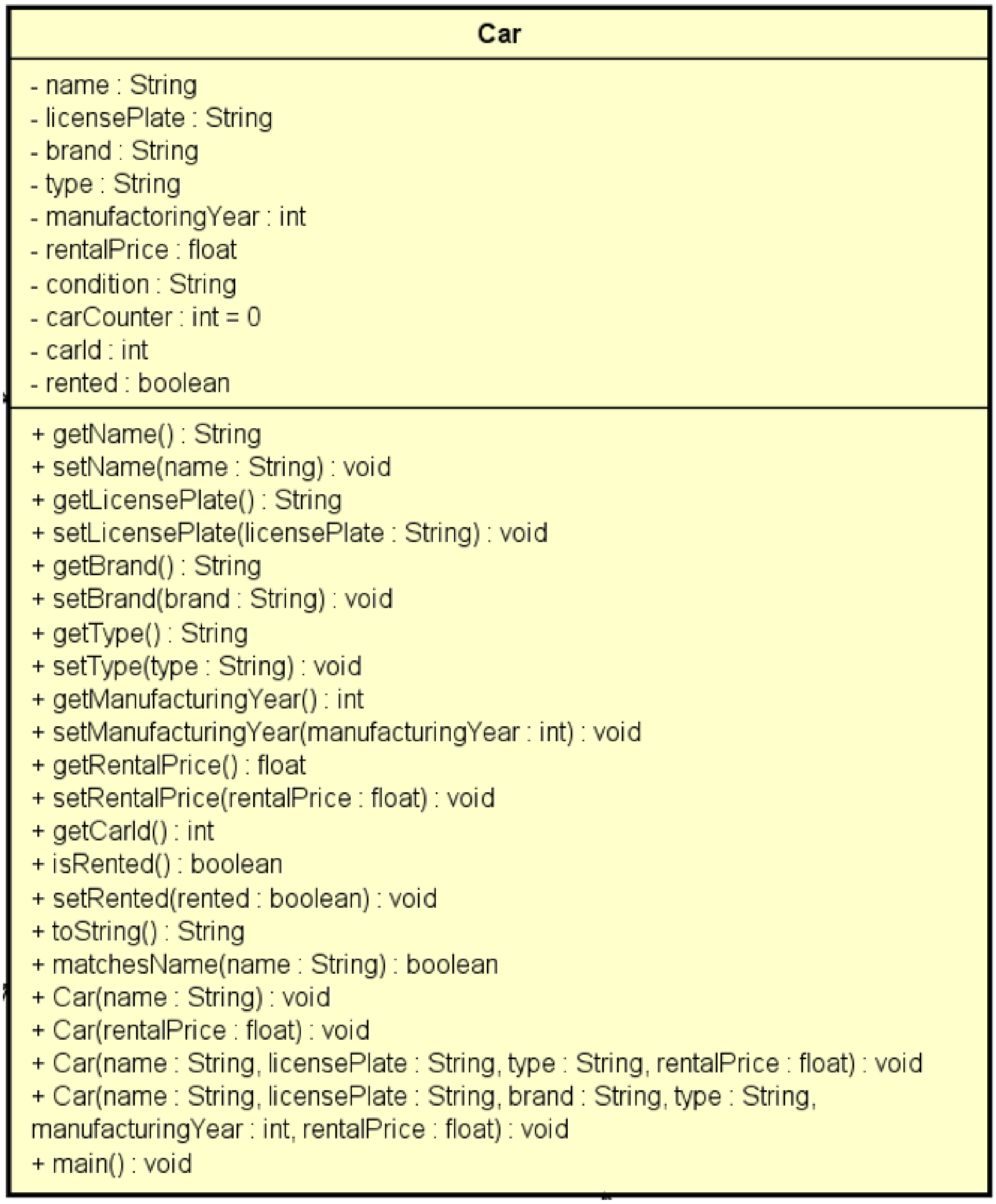
+ CarOwner(username, password, name, phone): Constructor.
+ getName(): Retrieves the owner's name.
+ loadOwnedCarsFromFile() and saveOwnedCarsToFile(): Loads and saves car data to a file.
- viewOwnedCars(): view car data from file
- addCarToStore() and removeCarFromStore(): Adds/removes cars in the store inventory.
- addCarToOwnedListAndStore(): Adds a car to the customer's owned car list and the store’s inventory.

- saveCarToStore(): Saves a car to the store inventory for renting or selling.
- removeCarFromStoreFile(): Removes a car from the store inventory file.
- saveOwnedCarsToFile(): Saves all cars owned by the car owner to a file.

OOP Technique: Inheritance

- Inherits from **Person** for shared functionality.
- **Composition** is used to interact with **Store** and **Car** classes.

3.7. Car



Car

- name : String
- licensePlate : String
- brand : String
- type : String
- manufactoringYear : int
- rentalPrice : float
- condition : String
- carCounter : int = 0
- carId : int
- rented : boolean

+ getName() : String
+ setName(name : String) : void
+ getLicensePlate() : String
+ setLicensePlate(licensePlate : String) : void
+ getBrand() : String
+ setBrand(brand : String) : void
+ getType() : String
+ setType(type : String) : void
+ getManufacturingYear() : int
+ setManufacturingYear(manufacturingYear : int) : void
+ getRentalPrice() : float
+ setRentalPrice(rentalPrice : float) : void
+ getCarId() : int
+ isRented() : boolean
+ setRented(rented : boolean) : void
+ toString() : String
+ matchesName(name : String) : boolean
+ Car(name : String) : void
+ Car(rentalPrice : float) : void
+ Car(name : String, licensePlate : String, type : String, rentalPrice : float) : void
+ Car(name : String, licensePlate : String, brand : String, type : String, manufacturingYear : int, rentalPrice : float) : void
+ main() : void

Purpose: Represents a car with its detailed properties (e.g., brand, price, license plate).

Attributes:

- name: Name of the car.

- licensePlate: Unique car identifier.

- brand: Manufacturer or brand.

- type: Type of the car.

- manufacturingYear: The year the car was built.

- rentalPrice: The price for renting the car.

- condition: Current condition of the car.

- isRented: Boolean flag to check rental status.

Methods:

(+) Getters/Setters: getName(), setName(), getRentalPrice(), etc.

+ isRented() and setRentalStatus(): Manage rental state.

+ toString(): Converts car details to a string format.

+ matchesName(): Saves all cars owned by the car owner to a file.

+ Car(name: String): Constructor to initialize a car with a name.

+ Car(rentalPrice: float): Constructor to initialize a car with a rental price.

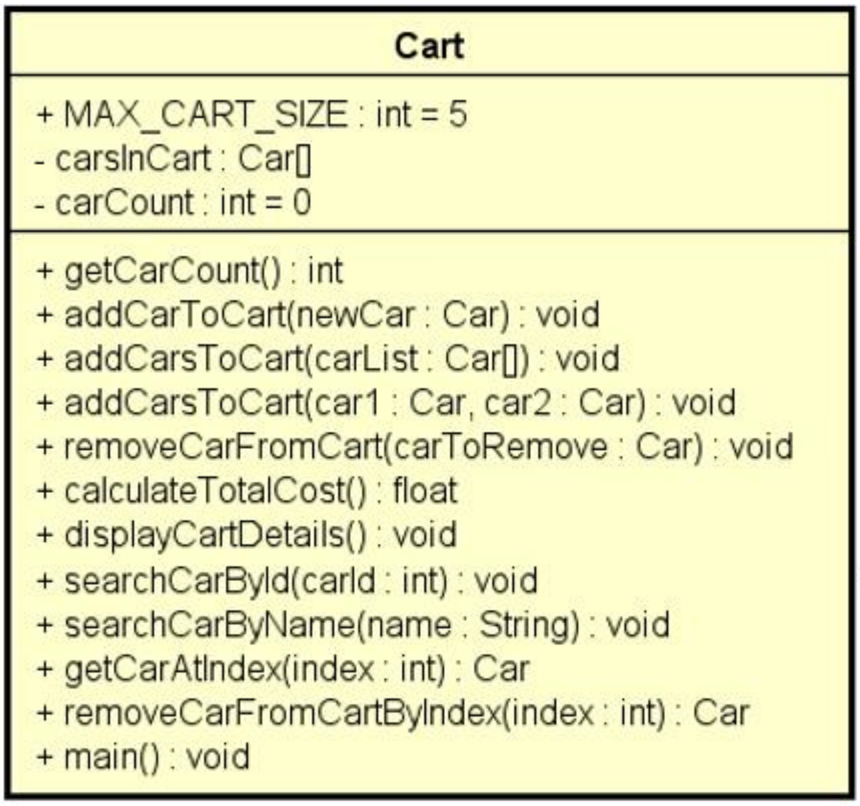
+ Car(name: String, licensePlate: String, type: String, rentalPrice: float): This constructor initializes a **Car object** with basic information: its name, license plate, type, and rental price.

+ Car(name: String, licensePlate: String, brand: String, type: String, manufacturingYear: int, rentalPrice: float): This constructor provides a detailed initialization for a Car object, including additional information such as brand and manufacturing year.

OOP Technique: Encapsulation

- Private attributes are accessed/modified via public getters and setters.

3.8. Cart


Cart
+ MAX_CART_SIZE : int = 5
- carsInCart : Car[]
- carCount : int = 0
+ getCarCount() : int
+ addCarToCart(newCar : Car) : void
+ addCarsToCart(carList : Car[]) : void
+ addCarsToCart(car1 : Car, car2 : Car) : void
+ removeCarFromCart(carToRemove : Car) : void
+ calculateTotalCost() : float
+ displayCartDetails() : void
+ searchCarById(carId : int) : void
+ searchCarByName(name : String) : void
+ getCarAtIndex(index : int) : Car
+ removeCarFromCartByIndex(index : int) : Car
+ main() : void

Purpose: Manages a temporary collection of cars added by customers for rental purposes.

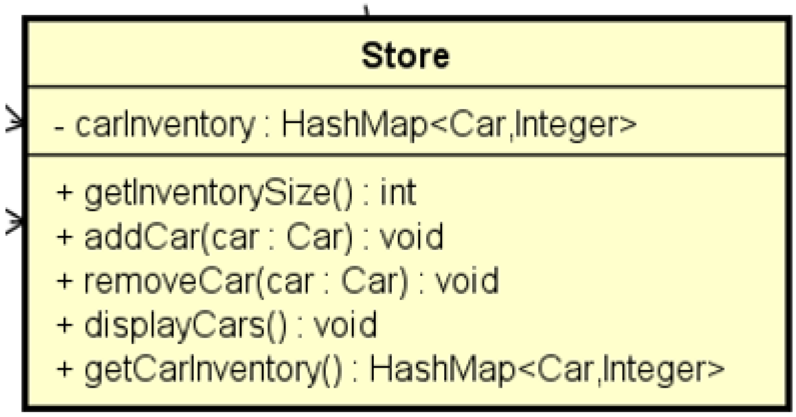
Attributes:

+ MAX_CART_SIZE: A constant to limit the number of cars in the cart.
- carsInCart: A list to hold cars added to the cart.
- carCount: Tracks the number of cars currently in the cart.

Methods:

+ getCarCount(): Retrieves the current number of cars in the cart.
+ addCarToCart(): Adds a car to the cart.
+ removeCarFromCart(): Removes a car.
+ calculateTotalCost(): Computes the total rental price.
+ displayCartDetails(): Displays cars in the cart.
+ searchCarById(carId: int): Searches for a car in the cart using its ID.
+ seachCarbyName(name: String): Searches for a car by its name in the cart.
+ getCarAtIndex(index: int): Retrieves a car at a specific index in the cart.
+ removeCarFromCartByIndex(index: int): Removes a car from the cart using its index.

3.9. Store



Store

- carInventory : HashMap<Car,Integer>

+ getInventorySize() : int
+ addCar(car : Car) : void
+ removeCar(car : Car) : void
+ displayCars() : void
+ getCarInventory() : HashMap<Car,Integer>

Purpose: Represents a car inventory system and manages car availability.

Attributes:

- carInventory: A HashMap<Car, Integer> to map cars to their quantities.

Methods:

+ addCar(): Adds a car to the inventory.
+ removeCar(): Removes a car from the inventory.
+ displayCars(): Displays all available cars.
+ getInventorySize(): Returns the size of the car inventory.
+ getCarInventory(): Retrieves the inventory as a data structure.

OOP Technique: **Composition**

- The **Store** class has a "Has-a" relationship with **Car**, meaning it contains and manages car objects.

4. CONCLUSION

The **Car Rental System** is a comprehensive and well-structured solution for managing car rental operations efficiently.

The class diagram effectively demonstrates key OOP techniques:

Encapsulation: Private attributes with public getters/setters.

Inheritance: The abstract Person class is extended by Admin, Customer, and CarOwner.

Abstraction: Common methods and attributes are abstracted in the Person class.

Polymorphism: Overriding shared methods like displayMenu() in subclasses.

Relationships: Associations, compositions, and aggregations between classes.

With features like cart management, data persistence, and rental status tracking, the system eliminates manual processes, reduces errors, and improves the overall user experience. It serves as a reliable platform for car rental businesses, enabling seamless interaction between stakeholders while ensuring effective car inventory management.

In summary, the **Car Rental System** successfully addresses the challenges of traditional car rental operations, offering a modern, automated, and user-friendly solution for managing rentals and inventory.